GRAPH CONTRASTIVE LEARNING WITH MODEL PERTURBATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph contrastive learning (GCL) has achieved great success in pre-training graph neural networks (GNN) without ground-truth labels. The performance of GCL mainly rely on designing high quality contrastive views via data augmentation. However, finding desirable augmentations is difficult and requires cumbersome efforts due to the diverse modalities in graph data. In this work, we study model perturbation to perform efficient contrastive learning on graphs without using data augmentation. Instead of searching for the optimal combination among perturbing nodes, edges or attributes, we propose to conduct perturbation on the model architectures (i.e., GNNs). However, it is non-trivial to achieve effective perturbations on GNN models without performance dropping compared with its data augmentation counterparts. This is because data augmentation 1) makes complex perturbation in the graph space, so it is hard to mimic its effect in the model parameter space with a fixed noise distribution, and 2) has different disturbances even on the same nodes between two views owning to the randomness. Motivated by this, we propose a novel model perturbation framework – PERTURBGCL to pre-train GNN encoders. We focus on perturbing two key operations in a GNN, including message propagation and transformation. Specifically, we propose *weightPrune* to create a dynamic perturbed model to contrast with the target one by pruning its transformation weights according to their magnitudes. Contrasting the two models will lead to adaptive mining of the perturbation distribution from the data. Furthermore, we present *randMP* to disturb the steps of message propagation in two contrastive models. By randomly choosing the propagation steps during training, it helps to increase local variances of nodes between the contrastive views. Despite the simplicity, coupling the two strategies together enable us to perform effective contrastive learning on graphs with model perturbation. We conduct extensive experiments on 15 benchmarks. The results demonstrate the superiority of PERTURBGCL: it can achieve competitive results against strong baselines across both node-level and graph-level tasks, while requiring shorter computation time. The code is available at `https://anonymous.4open.science/r/PerturbGCL-F17D`.

1 INTRODUCTION

Graph neural networks (GNN) (Kipf & Welling, 2016a; Hamilton et al., 2017; Gilmer et al., 2017) have become the *de facto* standard to model graph-structured data, such as social networks (Li & Goldwasser, 2019), molecules (Duvenaud et al., 2015), and knowledge graphs (Arora, 2020). Nevertheless, GNNs require task-specific labels to supervise the training, which is impractical in many scenarios where annotating graphs is challenging and expensive (Sun et al., 2019). Therefore, increasing efforts (Hou et al., 2022; Veličković et al., 2018; Hassani & Khasahmadi, 2020; Thakoor et al., 2022) have been made to train GNNs in an unsupervised fashion, so that the pre-trained model or learned representations can be directly applied to different downstream tasks.

Recently, graph contrastive learning (GCL) becomes the state-of-the-art approach for both graph-level (You et al., 2020; 2021; Suresh et al., 2021; Xu et al., 2021) and node-level (Qiu et al., 2020; Zhu et al., 2021b; Bielak et al., 2021; Thakoor et al., 2022) tasks. The general idea of GCL is to create two views of the original input using data augmentation (Jin et al., 2020), and then encode them with two GNN branches that share the same architectures and weights (You et al., 2020). Then, the model is optimized to maximize the mutual information between the two encoded representations according to contrastive objectives, such as InfoNCE (Oord et al., 2018) or Barlow Twins (Zbontar et al., 2021). As such, the performance of GCL mainly relies on designing high quality contrastive views (Zhang et al., 2021). Recently, intensive studies (You et al., 2020; Jin et al., 2020; Han et al., 2022) has been devoted to exploring effective augmentation strategies for graph data.

Despite their success, finding desirable augmentations requires cumbersome efforts, since the optimal augmentations are domain-specific and vary from graph to graph (You et al., 2020; Yin et al., 2022). To tackle this problem, SimGRACE (Xia et al., 2022) introduced the idea of model perturbation. Instead of searching for the optimal combination among perturbing nodes, edges or attributes in the graph space, SimGRACE conducts perturbation in *a unified parameter space* by adding Gaussian noise to model weights. However, we observe that SimGRACE may lead to sub-optimal representations compared with its data augmentation counterparts because of two reasons. *Firstly*, the data augmentation in the graph space is rather complicated and beyond Gaussian distribution. As a result, the weight perturbation based on Gaussian noises cannot achieve similar effects as data perturbation on representation learning (as illustrated in Section 2.2) . *Secondly*, the weight perturbation does not consider local variances among different nodes in a graph, since the perturbation is data-agnostic. Therefore, it still remains an important yet unsolved challenge to develop effective model perturbation framework for GCL, so that it can produce effective representations on both node and graph learning tasks in a more efficient manner.

To tackle these challenges, in this work, we propose a novel framework – PerturbGCL to train GNN encoders via model perturbation. Different from SimGRACE (Xia et al., 2022) that only focuses on weight perturbation, we make one step further to disturb the message passing (MP) of GNNs, since it allows to provide local disturbances between contrastive views. Specifically, we present *weightPrune* to construct a perturbed model by pruning the transformation weights of the target one. Unlike the Gaussian noise in SimGRACE (Xia et al., 2022), the pruned model will co-evolve with the target GNNs, leading to an adaptive mining of the noise perturbation from the data, i.e., data-driven. Furthermore, we propose *randMP* to offer local disturbances on nodes among contrastive views. It works by conducting k times of message propagation steps in each contrastive model, where k is randomly sampled on-the-fly. Informally, performing MP k times can be thought of as conducting convolution on the anchor node's k-hops of neighbors (Gao et al., 2018). On this basis, we can learn diverged but correlated representations from the two contrastive models with different k values due to the homophily theory (Altenburger & Ugander, 2018). Coupling the two strategies together yields a principled model perturbation solution tailored for GCL, whose effectiveness and efficiency have been empirically verified through our extensive experiments. We summarize our **main contributions** as follows:

- We introduce Perturbed Graph Contrastive Learning (PerturbGCL), a principled contrastive learning method on graphs that works by perturbing GNN architectures. PerturbGCL is flexible and easy to implement. To the best of our knowledge, PerturbGCL is the first model perturbation work that can achieve promising results on both node and graph learning tasks.
- PerturbGCL innovates to perturb GNN architectures from both the message passing and model weight perspectives via two effective perturbation strategies: *randMP* and *weightPrune*. By applying the two strategies jointly in contrastive models, PerturbGCL can be learned to mimic the effect of data augmentation from the model perturbation aspect.
- Extensive experiments across 15 benchmark datasets demonstrate the superiority of our proposal. Specifically, PerturbGCL can outperform state-of-the-art baselines without using data augmentation across two evaluation scenarios. Moreover, PerturbGCL is easy to optimize and runs generally faster than the strong GCL baselines.

2 METHODOLOGY

2.1 NOTATIONS AND PRELIMINARIES

Notations. Let $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$ be an undirected graph, where $\mathcal{V}$ is the set of nodes and $\mathcal{E}$ is the set of edges. $\mathbf{X} \in \mathbb{R}^{|\mathcal{V}| \times F}$ is the node feature matrix where the i-th row of $\mathbf{X}$ denote the F-dimensional feature vector of the i-th node in $\mathcal{V}$. We use f_w denote the mapping function that encodes each node $v \in \mathcal{G}$ into a D-dimensional representation $\mathbf{h}_v \in \mathbb{R}^D$.

Graph Neural Networks. To learn representations on graph data, we use graph neural networks (GNN) (Kipf & Welling, 2016a; Hamilton et al., 2017; Gilmer et al., 2017) as the encoder f_w. Without loss of generality, we present GNN as a message passing network:

$$\mathbf{h}_v^{(l)} = \sigma(\mathbf{a}_v^{(l-1)}\mathbf{W}^l), \;\; \mathbf{a}_v^{(l-1)} = g^{(l-1)}(\mathbf{h}_v^{(l-1)}, \{\mathbf{h}_u^{(l-1)} : u \in \mathcal{N}_v\}), \tag{1}$$

where $\mathbf{h}_v^{(l)} \in \mathbb{R}^D$ is the intermediate representation of node v at the l-the layer, $\mathcal{N}_v$ denotes the direct neighbors of node v. We use g to denote the message propagation (MP) function, which updates node representations by integrating its neighbors with a transformation function (Wu et al., 2019). $\mathbf{W}^l \in \mathbb{R}^{D\times D}$ is the transformation weight matrix and σ is the activation function, such as ReLU.

We often use the final layer's output as node-level representations, i.e., $\mathbf{h}_v = \mathbf{h}_v^L$ where L is the number of layers in a GNN. To get the graph-level representation $\mathbf{h}_\mathcal{G} \in \mathbb{R}^D$ for graph $\mathcal{G}$, we further aggregate all node-level representations in a graph via a readout function:

$$\mathbf{h}_\mathcal{G} = \text{READOUT}(\{\mathbf{h}_v^L : v \in \mathcal{V}\}), \;\; \mathbf{z}_\mathcal{G} = h(\mathbf{h}_\mathcal{G}) = \text{MLP}(\mathbf{h}_\mathcal{G}). \tag{2}$$

Here READOUT$(\cdot)$ can be the simple average pooling function or more sophisticated ones (Ying et al., 2018; Gao & Ji, 2019). $h(\cdot)$ is the projection head, and $\mathbf{z}_\mathcal{G} \in \mathbb{R}^D$ denotes the embedding towards loss estimation. In the development of our method, we follow the existing graph contrastive learning practices and consider three state-of-the-art GNNs: GCN (Kipf & Welling, 2016a), GIN (Xu et al., 2018), and ResGCN (Chen et al., 2019).

Graph Contrastive Learning. Graph contrastive learning (GCL) (You et al., 2020) has become an state-of-the-art approach for pre-training GNNs without ground-truth labels (Liu et al., 2021b;a). Unlike reconstruction-based methods (Perozzi et al., 2014; Kipf & Welling, 2016b), GCL is built upon a contrastive objective between the so-called positive pairs and negative pairs generated from the original data. Formally, given an anchor node v, let $(\mathbf{z}_v, \mathbf{z}_v^+)$ denote the representations of positive pairs and $(\mathbf{z}_v, \mathbf{z}_v^-)$ be the negative pairs. The contrastive loss could be defined as:

$$\mathcal{L}_{CL} = \frac{1}{|V|}\sum_{v\in\mathcal{V}} -\log \frac{\exp(\text{sim}(\mathbf{z}_v, \mathbf{z}_v^+)/\tau)}{\exp(\text{sim}(\mathbf{z}_v, \mathbf{z}_v^+)/\tau) + \sum_{u\in\mathcal{V}, u\neq v}\exp(\text{sim}(\mathbf{z}_v, \mathbf{z}_u^-)/\tau)}, \tag{3}$$

where τ is the temperature parameter, sim$(\cdot,\cdot)$ denotes the similarity function. By minimizing Eq. 3, the GNN encoder will be trained to enforce the similarity of the positive pairs while enlarging the distance of negative pairs in the hidden space. It is also worth noting that some GCL variants (Thakoor et al., 2022; Bielak et al., 2022) do not rely on negative samples. The key question in contrastive learning is how to generate effective positive (or negative) pairs. To this end, *graph augmentation* has been adopted as the golden rule in GCL (You et al., 2020; Jin et al., 2020). Typical graph augmentation techniques include edge perturbation, node masking and attribute masking.

2.2 The Proposed PerturbGCL Framework

$\triangleright$ **Motivation: How to perform model perturbation on GCL?** Given a graph $\mathcal{G} = (\mathbf{A}, \mathbf{X})$ where $\mathbf{A}$ is the adjacency matrix. For illustration purposes, we consider the mapping function $f_w(\mathbf{A}, \mathbf{X}; \mathbf{W})$ with one simple GCN (Wu et al., 2019) layer without activation function. Then, the hidden representation is computed by $f_w(\mathbf{A}, \mathbf{X}; \mathbf{W}) = g(\mathbf{A}, \mathbf{X})\mathbf{W}$, where $\mathbf{W}$ is the weight matrix and $g(\cdot,\cdot)$ is the message propagation operation defined in Eq. 1. $g(\mathbf{A}, \mathbf{X}) = \widetilde{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{A}+\mathbf{I}_\mathcal{G})\widetilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}_v$, where $\widetilde{\mathbf{D}}_{ii} = \sum_j(\mathbf{A}+\mathbf{I}_\mathcal{G})_{ij}$ is the degree matrix and $\mathbf{I}_\mathcal{G}$ is the identity matrix. **An intuitive solution** is adding Gaussian noise to the model weight and derive two contrastive views $f'_w(\mathbf{A}, \mathbf{X}; \mathbf{W})$ and $f''_w(\mathbf{A}, \mathbf{X}; \mathbf{W})$, as done in SimGRACE (Xia et al., 2022):

$$\begin{aligned} f'_w(\mathbf{A}, \mathbf{X}; \mathbf{W}) &= g(\mathbf{A}, \mathbf{X})\mathbf{W}, \\ f''_w(\mathbf{A}, \mathbf{X}; p(\mathbf{W})) &= g(\mathbf{A}, \mathbf{X})p(\mathbf{W}), \end{aligned} \tag{4}$$

where $p(\mathbf{W}) = \mathbf{W} + \eta\Delta_W$ is the perturbation function on model weight, and $\Delta_W \sim \mathcal{N}(0, \delta^2)$ represents the noise term sampled from Gaussian distribution with zero mean and variance δ^2, and η is a hyperparameter to scale the magnitude of the perturbation. Since the learning task is to minimize the distance between the two views, SimGRACE trains the model so that it is **robust to the Gaussian noise in the weight**.

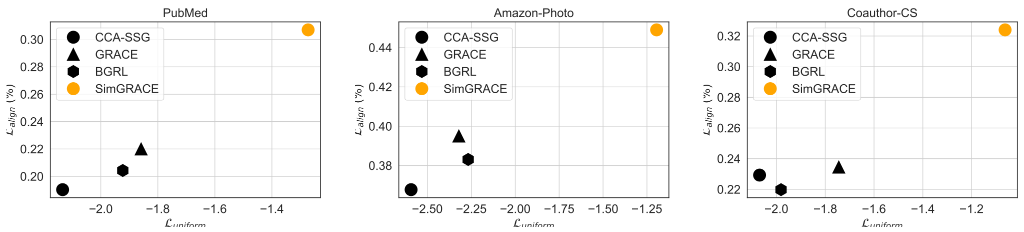

Figure 1: The *alignment* and *uniformity* performance (↓) of SimGRACE and augmentation-based GCL method (Thakoor et al., 2022) on the same perturbed graphs. *Black circles* (●) indicate the results of baselines. *Orange circles* (●) represent the results of SimGRACE. SimGRACE performs worse than standard GCL methods in modeling common perturbations achieved by data augmentation. Detailed experimental setups are listed in Appendix B.

However, we found that the contrastive views created by SimGRACE could be sub-optimal compared with its data-augmentation counterparts (see Figure 1). To measure the quality of representations learned by GCL models, we consider two popular metrics: *alignment* and *uniformity* (Wang & Isola, 2020) expressed as:

$$\begin{aligned} \mathcal{L}_{\text{align}}(f_w;\alpha) &\triangleq \mathbb{E}_{(x,y)\sim P_{\text{pos}}}[||\mathbf{h}_x - \mathbf{h}_y||_2^{\alpha}],\ \alpha > 0 \\ \mathcal{L}_{\text{uniform}}(f_w;t) &\triangleq \log \mathbb{E}_{(x,y)\overset{\text{i.i.d}}{\sim} P_{\text{data}}}[e^{-t||\mathbf{h}_x - \mathbf{h}_y||_2^2}].\ t > 0 \end{aligned} \tag{5}$$

P_{pos} is the distribution of positive pairs, i.e., augmentations of the same sample, P_{data} is the data distribution. $\mathcal{L}_{\text{align}}$ is used to measure if positive samples stay close in the hidden space. $\mathcal{L}_{\text{uniform}}$ is used to measure if random samples are scattered on the hypersphere of hidden space. In our experiments, we set $\alpha = t = 2$ following (Xia et al., 2022).

Figure 1 reports the test results of SimGRACE and augmentation-based GCL methods (Zhang et al., 2021; Thakoor et al., 2022; Zhu et al., 2020) on the same perturbed graphs created by random edge and attribute masking. Black circles (●) and rrange circles (●) represent the performance of baselines and SimGRACE, respectively. As can be observed, SimGRACE performs worse than three baselines across different datasets with a great margin. These results shed light on the bottleneck of SimGRACE in capturing common perturbations achieved by data augmentation. This is because SimGRACE is limited by the Gaussian noise and cannot handle perturbations in graph space. We thus ask: Can we design more advanced model perturbation strategies so that they can achieve similar functions as data augmentation done in GCL?

What is behind data augmentation in GCL? To answer this question, we start by analyzing the working mechanism behind the standard GCL framework. Let $T(\cdot)$ and $q(\cdot)$ denote augmentation functions on topology structure and node attributes, respectively. The two contrastive representations $f'_w(\mathbf{A}, \mathbf{X}; \mathbf{W})$ and $f''_w(\mathbf{A}, \mathbf{X}; \mathbf{W})$ of standard GCL (You et al., 2020) are defined as:

$$\begin{aligned} f'_w(\mathbf{A}, \mathbf{X}; \mathbf{W}) &= g(T'(\mathbf{A}), q'(\mathbf{X}))\mathbf{W}, \\ f''_w(\mathbf{A}, \mathbf{X}; \mathbf{W}) &= g(T''(\mathbf{A}), q''(\mathbf{X}))\mathbf{W}. \end{aligned} \tag{6}$$

That is, standard GCL framework learns to be robust to small disturbances (created by $(T'(\cdot), q'(\cdot))$ and $((T''(\cdot), q''(\cdot)))$ on the graph. We can easily observe two major properties of data augmentation as follows. ❶ It can disturb different nodes in a graph (different graphs in a graph set) differently. This is because $T(\cdot)$ and $q(\cdot)$ are random functions, such as edge masking, so they can have distinct effects even for the same input. ❷ The perturbation distribution incurred by data augmentation (e.g., $(T'(\cdot), q'(\cdot))$) is complicated and often beyond Gaussian noise. As analyzed in Figure 1, although SimGRACE is trained to be robust to Gaussian noise to some extent, it cannot handle the perturbation in the graph space well. These properties motivate us to design tailored model perturbation strategies from the two aspects.

▷ **Our PerturbGCL proposal.** To birdge the gap, we propose a novel model perturbation framework called PerturbGCL. Figure 2 provide the overview our framework. In this work, we build PerturbGCL upon the GraphCL pipeline (You et al., 2020), and follows its major components, such as two GNN branches and a non-linear projection head. The **main difference** is that GraphCL augments the input graph to get two views and then process them with two branches that share the same GNN architecture and weights, while PerturbGCL processes the original input graph with two non-symmetric GNN branches. One branch uses the original GNN model f_w, while the other branch

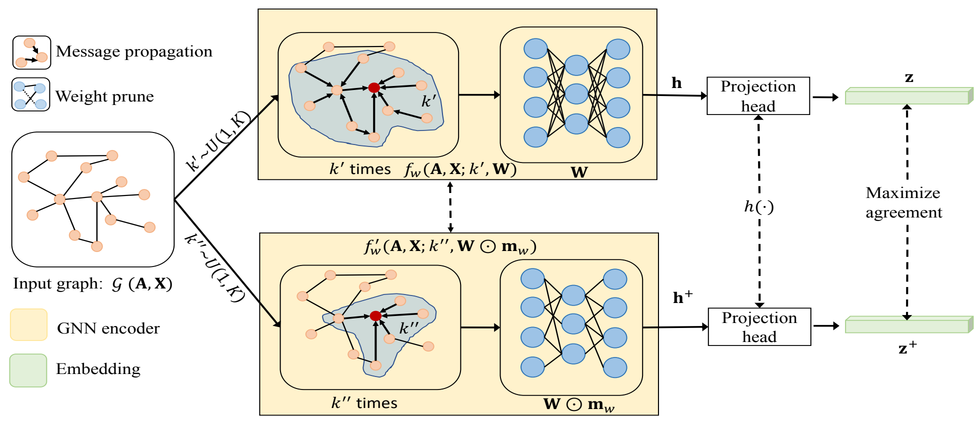

Figure 2: The overview of the proposed PerturbGCL framework. The original graph is fed into two asymmetric GNN branches: one is the target encoder f_w to be trained, and the other is the perturbed version f'_w that is pruned from the former online. The two branches share weights for their non-pruned parameters. Either branch has independent message propagation (MP) operations perturbed by a random number, i.e., k, to disturb nodes locally. Since the pruned branch is always obtained and updated from the latest target model, the two branches will co-evolve during training.

disturbs the message propagation process and model weights of f_w. We conduct perturbation on two major GNN operations, including message passing (MP) and transformation. On this basis, we introduce the following simple yet effective perturbation strategies: *randMP* and *weightPrune*, towards effective model augmentation for GNN architectures.

Strategy #1: *weightPrune*. Recently, model pruning has attracted increasing attention for model compression thanks to the popularity of the lottery ticket hypothesis (Frankle & Carbin, 2018). Work in (Chen et al., 2021) found that GNN can be pruned to a sparse sub-network without significant performance drop using rewinding techniques. These observations indicate the latent of pruning as a practical perturbation approach to GNN's weights. Inspired by this, we propose *weightPrune*, which creates the perturbed branch by pruning the model parameters of the target encoder. Specifically, assuming $\mathbf{W}$ denote the weights of the target branch, $\mathbf{m}_w$ be the mask of the pruned branch, which has the same size as $\mathbf{W}$. At each iteration, we prune the target branch according to a pre-defined prune ratio s according to the magnitude of weight values, i.e., masking weights out if their magnitudes are ranked below s. By changing s, we can control the distortion degree of the target branch to a certain extent. After that, the target and perturbed branches will use $\mathbf{W}$ and $\mathbf{W} \odot \mathbf{m}_w$ respectively as model weights to generate representations, which are then fed into the contrastive loss. Since the mask indicator $\mathbf{m}_w$ is continuously updated from the latest target model, the two branches will co-evolve during training.

Strategy #2: *randMP*. Message passing is another critical component of the GNN architecture, since it offers the flexibility to aggregate features from multi-hop neighbors. Performing MP for k times over the graph $\mathcal{G}$ is equivalent to updating node v's representation based on its k-hop local subgraph. In light of this, k could be naturally regarded as a perturbation factor, where different k values generate *diverse but semantically correlated representations* for the same node. To implement randMP, we will randomly sample two k values at each iteration: one is for the target branch (i.e., k') and the other for the perturbed branch (i.e., k''). Formally, if we assume $g(\mathcal{A}, \mathbf{X}) = \widetilde{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{A} + \mathbf{I}_{\mathcal{G}})\widetilde{\mathbf{D}}^{-\frac{1}{2}}\mathbf{X}_v$, performing MP k times gives $g(\mathcal{A}, \mathbf{X})^k = (\widetilde{\mathbf{D}}^{-\frac{1}{2}}(\mathbf{A} + \mathbf{I}_{\mathcal{G}})\widetilde{\mathbf{D}}^{-\frac{1}{2}})^k\mathbf{X}_v$. In experiments, we consider sampling k during the training because it may enforce the GNN encoder to learn generable representations invariant to different combinations of local enclosing graphs.

To sum up, different from existing methods (Xia et al., 2022; Thakoor et al., 2022) that disturb model weights with Gaussian noise, we suggest a principled approach to effectively perturb GNN architectures from their message propagation and feature transformation perspectives, via two simple perturbation techniques: *randMP* and *weightPrune*. *randMP* aims to map the same input graph into two semantically similar representations by conducting a random number of message passing steps. Meanwhile, *weightPrune* targets to increase the diversity of two representations via model pruning. Combining the two strategies enables us to spot the sweet point between the two view representations (Tian et al., 2020), i.e., correlated but diverged enough. The complete optimization procedure of our model is outlined in Algorithm 1 and 2 in Appendix.

2.3 More Discussions on PerturbGCL

PerturbGCL is complementary to other GCL efforts. We focus on training GCL by only using model perturbation. It can be easily combined with existing contrastive learning advances, such as mature graph augmentation techniques (You et al., 2020) and the negative-sample free contrastive loss objectives (Bielak et al., 2022; Thakoor et al., 2022), as we will show in Section 3.4.

Computational complexity analysis. In addition to saving a lot of time in searching for the optimal data augmentation strategies, we analyze the complexity of PerturbGCL. Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ and the GNN encoder f_w. The time complexity for most popular GNN architectures (Kipf & Welling, 2016a; Veličković et al., 2017; Gilmer et al., 2017) is $\mathcal{O}(|\mathcal{E}| + |\mathcal{V}|)$, where $\mathcal{O}(|\mathcal{E}|)$ and $\mathcal{O}(|\mathcal{V}|)$ are mainly caused by the message propagation and feature transformation operations, respectively. PerturbGCL performs two encoder computations per update step (one for each GNN branch) plus a node-level projection head. Assuming that the backward pass to be approximately as costly as a forward pass and ignoring the cost for weight pruning as it is small and negligible. Thus the total time complexity per update step for PerturbGCL is $4C_{\text{encoder}}(K|\mathcal{E}| + |\mathcal{V}|s) + 2C_{\text{head}}(|\mathcal{V}|) + C_{\text{loss}}$, where $C_{\cdot}$ are constants depending on architecture of the different components, K is the maximum number of MP operations considered (e.g., $K = 3$), and s is the pruning ratio. It is worth noting that although our model at most takes K times MP operations in forward pass, due to weight pruning (e.g., $s = 70\%$), the computation costs for feature transformation and backpropagation are significantly lower than standard GCL methods. Therefore, the total running cost of PerturbGCL can be further accelerated in practice. We empirically analyze the efficiency of our model in Section 3.5.

3 Experiments

In this section, we evaluate the performance of PerturbGCL. Specifically, we first visualize the weight distribution and the alignment between positive pairs in Section 3.1 to investigate what the proposed two strategies actually do. Then we evaluate the effectiveness of PerturbGCL in node classification with several benchmarks and SOTA baselines in Section 3.2. Next, we test the performance of PerturbGCL in graph classification under both unsupervised and semi-supervised settings in Section 3.3. After that, we evaluate the contributions of different components in Section 3.4, as well as how it improves the efficiency of GCL against standard methods in Section 3.5. The experiment setting and more experiments are summarized in Appendices A and E. Through the experiments, the main observations are highlighted.

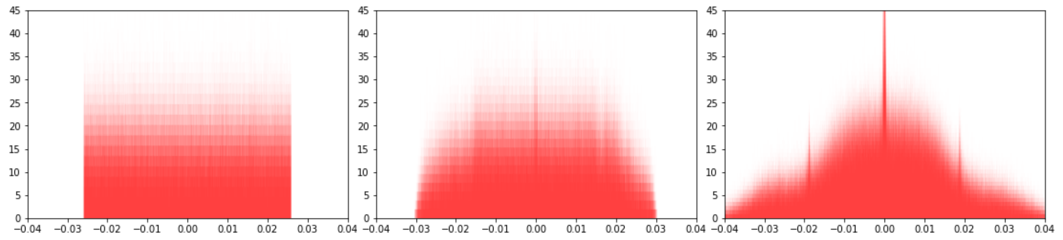

Figure 3: Visualization of weight distribution (from left to right: initial weights, PerturbGCL w/o. *weightPrune*, and PerturbGCL) on Coauthor-Phy. The x-axis indicates weight values and y-axis is the count. Obviously, the number of activated neurons after using *weightPrune* is significantly smaller than others. It shows that *weightPrune* can regularize the model.

3.1 What are *weightPrune* and *randMP* doing? A case study

We visualize the weight distribution of PeruturbGCL during training in Figure 3 and Figure 8 (in Appendix). It shows that ① **by continuously pruning the target model along the training, *weightPrune* can regularize the target model progressively.** From the weight histograms in Figure 3, we can see that the bars around zero become higher and higher, which indicates more neurons are inactivated in the end. This observation shows the regularization effect of *weightPrune*. Since effective

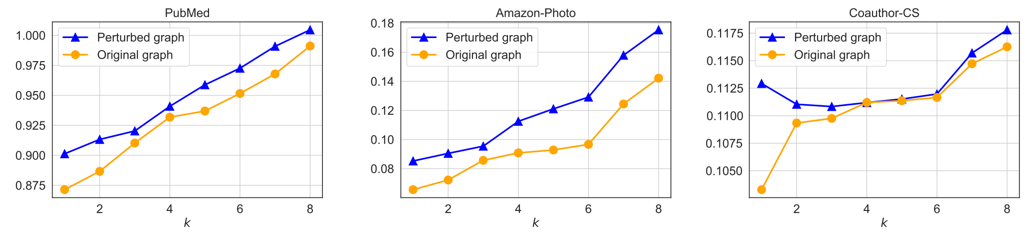

Figure 4: The visualization of PerturbGCL w.r.t. different k values on the original graphs and the perturbed graphs generated by data augmentation. The x-axis indicates propagation steps and y-axis is the $\mathcal{L}_{\text{align}} \downarrow$. The gap between the blue and orange lines indicate the generalization ability. Apparently, performing more MP steps will increase the diversity of two positive views since $\mathcal{L}_{\text{align}}$ increases. Sweet spots (i.e., minimum performance gap) exist across three scenarios.

Table 1: Test accuracy on benchmark datasets in terms of node classification. We report both mean accuracy and standard deviation. A.R. denotes the averaged rank.

	Method	Cora	PubMed	Compute	Photo	CS	Phy	A.R. ↓
Supervised	GCN	81.5	79.0	86.51 ± 0.54	92.42 ± 0.22	93.03 ± 0.31	95.65 ± 0.16	8.33
	GAT	83.0 ± 0.7	79.0 ± 0.3	86.93 ± 0.29	92.56 ± 0.35	92.31 ± 0.24	95.47 ± 0.15	8.00
Unsupervised	Raw Features	47.9 ± 0.4	69.1 ± 0.3	73.81 ± 0.00	78.53 ± 0.00	90.37 ± 0.00	93.58 ± 0.00	14.33
	DeepWalk	70.7 ± 0.6	74.3 ± 0.9	85.68 ± 0.06	89.44 ± 0.11	84.61 ± 0.22	91.77 ± 0.15	13.67
	GAE	71.5 ± 0.4	72.1 ± 0.5	85.27 ± 0.19	91.62 ± 0.13	90.01 ± 0.71	94.92 ± 0.07	12.50
	DGI	82.3 ± 0.6	76.8 ± 0.6	83.95 ± 0.47	91.61 ± 0.22	92.15 ± 0.63	94.51 ± 0.52	11.00
	MVGRL	83.5 ± 0.4	80.1 ± 0.7	87.52 ± 0.11	91.74 ± 0.07	92.11 ± 0.12	95.33 ± 0.03	7.50
	GRACE	81.9 ± 0.4	80.6 ± 0.4	$\mathbf{89.53 \pm 0.35}$	92.78 ± 0.45	91.12 ± 0.20	–	6.40
	GCA	83.4 ± 0.3	80.3 ± 0.4	87.85 ± 0.31	92.49 ± 0.09	93.10 ± 0.01	95.68 ± 0.05	5.33
	BGRL	81.8 ± 0.3	80.4 ± 0.5	$\mathbf{89.68 \pm 0.31}$	92.87 ± 0.27	93.21 ± 0.18	95.56 ± 0.12	4.83
	GBT	–	–	88.14 ± 0.33	92.63 ± 0.44	92.95 ± 0.17	95.07 ± 0.17	7.00
	InfoGCL	83.5 ± 0.3	79.1 ± 0.2	–	–	–	–	6.00
	CCA-SSG	$\mathbf{84.2 \pm 0.4}$	81.6 ± 0.4	88.74 ± 0.28	93.14 ± 0.14	93.31 ± 0.22	95.38 ± 0.06	3.17
	AFGRL	82.3 ± 0.4	79.7 ± 0.2	$\mathbf{89.88 \pm 0.33}$	93.22 ± 0.28	93.27 ± 0.17	95.69 ± 0.10	3.83
	SimGRACE	78.5 ± 0.3	79.3 ± 0.5	86.42 ± 0.35	91.55 ± 0.22	92.37 ± 0.33	94.37 ± 0.15	10.67
	PerturbGCL	83.3 ± 0.5	$\mathbf{82.10 \pm 0.37}$	88.45 ± 0.77	$\mathbf{93.62 \pm 0.40}$	$\mathbf{94.18 \pm 0.09}$	$\mathbf{95.85 \pm 0.08}$	2.33

regularization can improve the generalization ability of neural networks (Scholkopf & Smola, 2018), we believe that why the proposed *weightPrune* can improve the performance.

To investigate the effect of *randMP*, we report the impacts of different k values on PerturbGCL in terms of the alignment between positive views. From Figure 4, we observe that ② ***randMP* can improve the diversity of contrastive views when k increases, and sweet points widely exist across three datasets.** In Figure 4, with the increase of k, the generalization gap tends to first decrease to the sweet points and then increase a little bit. It validates the effect of *randMP* in generating correlated but diverged views. On the other hand, it indicates the potential of *randMP* to improve the generalization ability, i.e., these sweet points.

3.2 Can PerturbGCL perform well on Node classsification Task?

We first examine the effectiveness of PerturbGCL in node classification. Results of 15 baseline methods across 6 benchmark datasets are collected in Table 1. We make the following observations: ③ **PerturbGCL can achieve better node classification results than SOTA GCL methods without using data augmentation.** From Table 1, PerturbGCL gains 4 best performances among 6 evaluation scenarios. On average, it ranks 2.33 among 13 augmentation-based baselines including strong methods, such as BGRL and CCA-SSG, which indicates the power of model perturbation based contrastive learning. Meanwhile, ④ **PerturbGCL outperforms the model perturbation baseline – SimGRACE with great margins.** Among 6 datasets, SimGRACE loses to PerturbGCL in all cases. Specifically, PerturbGCL improves SimGRACE 6.11%, 3.53%, 2.34%, 2.26%, 1.95%, and 1.56% on Cora, PubMed, Computer, Photo, CS, and Phy, respectively. This result is in line with our analysis in Section 2.2.

3.3 Can PerturbGCL generalize well to graph classification task?

To validate the effectiveness of PerturbGCL on graph classification, we compare it with state-of-the-art graph-level GCL methods on different datasets. Table 2 and Table 9 (in Appendix) report

Table 2: Test accuracy on benchmark datasets in TUdatasets in terms of the unsupervised setting for graph classification. – means that results are not available in published papers.

Methods	NCI1	PROTEINS	DD	MUTAG	COLLAB	RDT-B	RDT-M5K	IMDB-B	A.R. ↓
GL	–	–	–	81.66 ± 2.11	–	77.34 ± 0.18	41.01 ± 0.17	65.87 ± 0.98	9.50
WL	$\mathbf{80.01 \pm 0.50}$	72.92 ± 0.56	–	80.72 ± 3.00	–	68.82 ± 0.41	46.06 ± 0.21	72.30 ± 3.44	7.50
DGK	$\mathbf{80.31 \pm 0.46}$	73.30 ± 0.82	–	87.44 ± 2.72	–	78.04 ± 0.39	41.27 ± 0.18	66.96 ± 0.56	6.50
node2vec	54.89 ± 1.61	57.49 ± 3.57	–	72.63 ± 10.20	–	–	–	–	10.67
sub2vec	52.84 ± 1.47	53.03 ± 5.55	–	61.05 ± 15.80	–	71.48 ± 0.41	36.68 ± 0.42	55.26 ± 1.54	11.50
graph2vec	73.22 ± 1.81	73.30 ± 2.05	–	83.15 ± 9.25	–	75.78 ± 1.03	47.86 ± 0.26	71.10 ± 0.54	8.00
MVGRL	–	–	–	75.40 ± 7.80	–	82.00 ± 1.10	–	63.60 ± 4.20	9.67
InfoGraph	76.20 ± 1.06	74.44 ± 0.31	72.85 ± 1.78	89.01 ± 1.13	70.65 ± 1.13	89.53 ± 0.84	$\mathbf{55.99 \pm 0.28}$	73.03 ± 0.87	3.63
GraphCL	77.87 ± 0.41	74.39 ± 0.45	$\mathbf{78.62 \pm 0.40}$	86.80 ± 1.34	71.36 ± 1.15	89.53 ± 0.84	$\mathbf{55.99 \pm 0.28}$	71.14 ± 0.44	4.13
JOAO	78.07 ± 0.47	74.55 ± 0.41	77.32 ± 0.54	87.35 ± 1.02	69.50 ± 0.36	85.29 ± 1.35	55.74 ± 0.63	70.21 ± 3.08	5.63
JOAOv2	78.36 ± 0.53	74.07 ± 1.10	77.40 ± 1.15	87.67 ± 0.79	69.33 ± 0.34	86.42 ± 1.45	56.03 ± 0.27	70.83 ± 0.25	4.75
SimGRACE	79.12 ± 0.44	75.35 ± 0.09	77.44 ± 1.11	89.01 ± 1.31	71.72 ± 0.82	$\mathbf{89.51 \pm 0.89}$	$\mathbf{55.91 \pm 0.34}$	71.30 ± 0.77	3.13
PerturbGCL	$\mathbf{80.24 \pm 0.45}$	$\mathbf{76.08 \pm 0.30}$	$\mathbf{78.33 \pm 0.37}$	$\mathbf{89.97 \pm 0.50}$	$\mathbf{75.06 \pm 0.87}$	88.98 ± 0.67	$\mathbf{55.78 \pm 0.72}$	$\mathbf{74.14 \pm 0.50}$	2.13

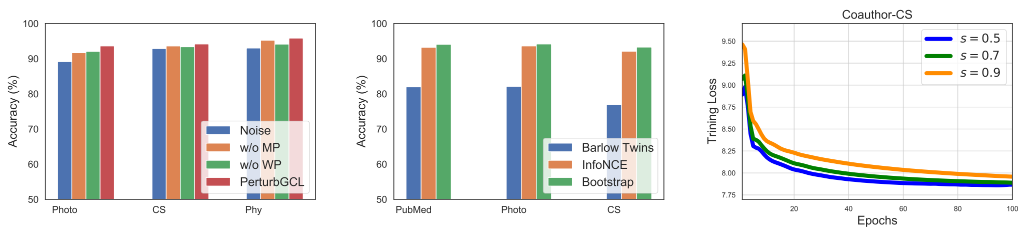

Figure 5: **Left:** Ablation study of PerturbGCL. **Middle:** The impact of different contrastive objectives. **Right:** Empirical training curves of PerturbGCL with different s values.

the results on unsueprvised and semi-supervised settings, respectively. We observed that ⑤ **PerturbGCL generally performs better than other baselines across two graph learning tasks.** From the unsupervised setting (See Table 2), PerturbGCL achieves the best (or comparable best) results on 6 of 8 datasets, and obtain substantial improvements on COLLAB and IMDB-B datasets. In the semi-supervised setting (See Table 9 in Appendix), PerturbGCL generally performs better than other baselines across 7 comparisons and always ranks top three on all the datasets. These results demonstrate the effectiveness of PerturbGCL on graph learning task.

3.4 Ablation Study

We investigate the contributions of different components in PerturbGCL. Figure 5 and Figure 9 in Appendix report the results on graph and node datasets, respectively. We observe that ⑥ **PerturbGCL benefits from the combination of randMP with weightPrune.** From the figures, PerturbGCL consistently outperforms two variants (i.e., w/o MP and w/o WP) in all cases, which indicates the reciprocal effects of using *randMP* and *weightPrune* together. Moreover, ⑦ **replacing weightPrune with Gaussian noise, PerturbGCL drops significantly.** In both node and graph scenarios, PerturbGCL outperforms the ”noise” variant with a great margin. It verifies the effectiveness of the proposed *weightPrune* strategy.

We also test the results of PerturbGCL under different contrastive objectives, such as Barlow Twins (Bielak et al., 2021), Bootstrap (Thakoor et al., 2022), and InfoNCE (You et al., 2020) (Please refer to Appendix C for details). From Figure 5 (middle), we observe that ⑧ **PerturbGCL performs generally better on InfoNCE and Barlow Twins objectives.** Given that InfoNCE is standard contrastive loss and Barlow Twins is negative-sample free, PerturbGCL is ready to be applied on scenarios with informative negative sample or without negative samples by using different losses.

3.5 Can PerturbGCL improve the training efficiency of GCL?

We compare the proposed PerturbGCL with strong GCL baselines in terms of the training costs in Table 3 and report the optimization curves in Figure 5 (right panel). Experimental configurations are listed in Appendix D. We observed that ⑨ **using pure model perturbation, PerturbGCL is training efficient**. From Table 3, we can see that PerturbGCL runs significantly faster per epoch than strong baselines in general, and the performance gap is particularly evident in graph datasets. Besides, PerturbGCL can converge within one hundred epochs in practice as shown in Figure 5 (Right). Thus, the total training time of PerturbGCL could be further reduced.

Table 3: Running time per epoch (in seconds). Baseline indicates BGRL and GraphCL for node and graph classification, respectively. All the methods are evaluated on GeForce RTX 2080 Ti GPUs.

	Node Benchmark				Graph Benchmark			
	PubMed	Computer	Photo	CS	NCI1	COLLAB	RDT-B	RDT-M2K
Baseline	0.14	0.16	0.08	0.21	4.02	10.84	38.35	80.79
PerturbGCL	0.10	0.09	0.05	0.17	1.42	3.01	6.21	12.80
Speedup (vs Baseline)	**1.40x**	**1.77x**	**1.60x**	**1.23x**	**2.83x**	**3.60x**	**6.17x**	**6.31x**

3.6 Further Analysis

We finally investigate the sensitivity of PerturbGCL w.r.t. the propagation degree K and prune ratio s in Figure 10 (Left) of Appendix, the impact of graph augmentation on PerturbGCL in Figure 10 (Middle), and the learning capacity of PerturbGCL in Figure 10 (Right). We can observe that ⑩ **PerturbGCL performs stably when** $K \in [1, 2, 3, 4, 5]$ **and** $s \in [0.7, 0.9]$. In Figure 10 left, the performance of PerturbGCL when $K = 0.9$ (or 0.7) is consistently better than others. **PerturbGCL is complementary with advanced graph augmentation.** From Figure 10 (Middle), by feeding the augmented graphs as input, PerturbGCL can be further improved. However, the trade-off is that the improvement is not huge but the time to search optimal augmentation strategies is costing. **Although PerturbGCL is trained based on original graphs, it can generalize to perturbed graphs well**. As shown in Figure 10 (Right), PerturbGCL has lower $\mathcal{L}_{\text{align}}$ and $\mathcal{L}_{\text{uniform}}$ values than SimGRACE and strong GCL baseline, which indicates the effectiveness of the proposed model.

4 Related Work

We briefly introduce some related graph contrastive learning methods (Xie et al., 2022) and refer readers to (Zhou et al., 2020) for a comprehensive review of graph neural networks.

Graph contrastive learning with data augmentation. Similar to contrastive learning on images (Chen et al., 2020), data augmentation is crucial to the success of contrastive learning on graphs (GCL). Recently there has been steady progress (You et al., 2020; 2021; Qiu et al., 2020; Lee et al., 2022; Luo et al., 2022) in designing or identifying informative augmentation strategies to boost the performance of GCL. GraphCL (You et al., 2020) introduces four augmentation prototypes for graphs, including node dropping, edge perturbation, attribute masking, and subgraph sampling. MoCL (Sun et al., 2021) and G-Mixup (Han et al., 2022) propose to utilize domain knowledge such as bioisosteres and graphon to aid augmentation. AutoGCL (Yin et al., 2022), JOAO (You et al., 2021), and GPA (Zhang et al., 2022) suggest to leverage extra AutoML techniques (Waring et al., 2020) to free human labor on augmentation choices. Unlike this line of research, we focus on training GCL methods without explicitly graph augmentation.

Graph contrastive learning without data augmentation. To eliminate the influence of graph augmentation on GCL, AFGRL (Lee et al., 2022) suggests sampling nodes that share similar semantic information in the hidden space as positive samples. However, it requires non-negligible clustering efforts to spot positive pairs in the learning process. SimGRACE (Xia et al., 2022) proposes to train GCL by disturbing the model weight using Gaussian noise. However, SimGRACE could limit applications beyond Gaussian distribution, as illustrated in Figure 1. In this work, we introduce a tailored model perturbation framework for GNN encoders without constraining the noise distribution.

5 Conclusion

In this work, we explore how to perform contrastive learning on graphs without using data augmentation, and propose a principled framework – PerturbGCL, which is built upon pure model perturbations. Specifically, motivated by the fact that GNN can be divided into message propagation and feature transformation operations, we develop two tailored perturbation strategies: *randMP* and *weightPrune*, to effectively disturb GNN's two crucial operations accordingly. We build connections between our model perturbation strategies and well-established graph augmentation techniques to understand the working mechanism of PerturbGCL. Through extensive experiments across multiple datasets and different graph learning tasks, we show that PerturbGCL can achieve competitive results against strong baselines, while requiring substantially shorter computation time for training.

REFERENCES

Bijaya Adhikari, Yao Zhang, Naren Ramakrishnan, and B Aditya Prakash. Sub2vec: Feature learning for subgraphs. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*, pp. 170–182. Springer, 2018.

Kristen M Altenburger and Johan Ugander. Monophily in social networks introduces similarity among friends-of-friends. *Nature human behaviour*, 2(4):284–290, 2018.

Siddhant Arora. A survey on graph neural networks for knowledge graph completion. *arXiv preprint arXiv:2007.12374*, 2020.

Piotr Bielak, Tomasz Kajdanowicz, and Nitesh V Chawla. Graph barlow twins: A self-supervised representation learning framework for graphs. *arXiv preprint arXiv:2106.02466*, 2021.

Piotr Bielak, Tomasz Kajdanowicz, and Nitesh V Chawla. Graph barlow twins: A self-supervised representation learning framework for graphs. *Knowledge-Based Systems*, pp. 109631, 2022.

Tianlong Chen, Yongduo Sui, Xuxi Chen, Aston Zhang, and Zhangyang Wang. A unified lottery ticket hypothesis for graph neural networks. In *ICML*, pp. 1695–1706. PMLR, 2021.

Ting Chen, Song Bian, and Yizhou Sun. Are powerful graph neural nets necessary? a dissection on graph classification. *arXiv preprint arXiv:1905.04579*, 2019.

Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. A simple framework for contrastive learning of visual representations. In *ICML*, pp. 1597–1607. PMLR, 2020.

David K Duvenaud, Dougal Maclaurin, Jorge Iparraguirre, Rafael Bombarell, Timothy Hirzel, Alán Aspuru-Guzik, and Ryan P Adams. Convolutional networks on graphs for learning molecular fingerprints. *Advances in neural information processing systems*, 28, 2015.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

Hongyang Gao and Shuiwang Ji. Graph u-nets. In *international conference on machine learning*, pp. 2083–2092. PMLR, 2019.

Hongyang Gao, Zhengyang Wang, and Shuiwang Ji. Large-scale learnable graph convolutional networks. In *KDD*, pp. 1416–1424, 2018.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pp. 1263–1272. PMLR, 2017.

Aditya Grover and Jure Leskovec. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 855–864, 2016.

William L Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NeurIPS*, pp. 1025–1035, 2017.

Xiaotian Han, Zhimeng Jiang, Ninghao Liu, and Xia Hu. G-mixup: Graph data augmentation for graph classification. *arXiv preprint arXiv:2202.07179*, 2022.

Kaveh Hassani and Amir Hosein Khasahmadi. Contrastive multi-view representation learning on graphs. In *International Conference on Machine Learning*, pp. 4116–4126. PMLR, 2020.

Zhenyu Hou, Xiao Liu, Yuxiao Dong, Chunjie Wang, Jie Tang, et al. Graphmae: Self-supervised masked graph autoencoders. *arXiv preprint arXiv:2205.10803*, 2022.

Weihua Hu, Bowen Liu, Joseph Gomes, Marinka Zitnik, Percy Liang, Vijay Pande, and Jure Leskovec. Strategies for pre-training graph neural networks. *arXiv preprint arXiv:1905.12265*, 2019.

Wei Jin, Tyler Derr, Haochen Liu, Yiqi Wang, Suhang Wang, Zitao Liu, and Jiliang Tang. Self-supervised learning on graphs: Deep insights and new direction. *arXiv preprint arXiv:2006.10141*, 2020.

Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016a.

Thomas N Kipf and Max Welling. Variational graph auto-encoders. *arXiv preprint arXiv:1611.07308*, 2016b.

Namkyeong Lee, Junseok Lee, and Chanyoung Park. Augmentation-free self-supervised learning on graphs. *arXiv preprint arXiv:2112.02472*, 2021.

Namkyeong Lee, Junseok Lee, and Chanyoung Park. Augmentation-free self-supervised learning on graphs. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 7372–7380, 2022.

Chang Li and Dan Goldwasser. Encoding social information with graph convolutional networks forpolitical perspective detection in news media. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pp. 2594–2604, 2019.

Xiao Liu, Fanjin Zhang, Zhenyu Hou, Li Mian, Zhaoyu Wang, Jing Zhang, and Jie Tang. Self-supervised learning: Generative or contrastive. *TKDE*, 2021a.

Yixin Liu, Shirui Pan, Ming Jin, Chuan Zhou, Feng Xia, and Philip S Yu. Graph self-supervised learning: A survey. *arXiv preprint arXiv:2103.00111*, 2021b.

Youzhi Luo, Michael McThrow, Wing Yee Au, Tao Komikado, Kanji Uchino, Koji Maruhash, and Shuiwang Ji. Automated data augmentations for graph classification. *arXiv preprint arXiv:2202.13248*, 2022.

Julian McAuley, Christopher Targett, Qinfeng Shi, and Anton Van Den Hengel. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th international ACM SIGIR conference on research and development in information retrieval*, pp. 43–52, 2015.

Christopher Morris, Nils M Kriege, Franka Bause, Kristian Kersting, Petra Mutzel, and Marion Neumann. Tudataset: A collection of benchmark datasets for learning with graphs. *arXiv preprint arXiv:2007.08663*, 2020.

Annamalai Narayanan, Mahinthan Chandramohan, Rajasekar Venkatesan, Lihui Chen, Yang Liu, and Shantanu Jaiswal. graph2vec: Learning distributed representations of graphs. *arXiv preprint arXiv:1707.05005*, 2017.

Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. Deepwalk: Online learning of social representations. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 701–710, 2014.

Jiezhong Qiu, Qibin Chen, Yuxiao Dong, Jing Zhang, Hongxia Yang, Ming Ding, Kuansan Wang, and Jie Tang. Gcc: Graph contrastive coding for graph neural network pre-training. In *KDD*, pp. 1150–1160, 2020.

Bernhard Scholkopf and Alexander J Smola. *Learning with kernels: support vector machines, regularization, optimization, and beyond.* MIT press, 2018.

Prithviraj Sen, Galileo Namata, Mustafa Bilgic, Lise Getoor, Brian Galligher, and Tina Eliassi-Rad. Collective classification in network data. *AI magazine*, 29(3):93–93, 2008.

Oleksandr Shchur, Maximilian Mumme, Aleksandar Bojchevski, and Stephan Günnemann. Pitfalls of graph neural network evaluation. *arXiv preprint arXiv:1811.05868*, 2018.

Nino Shervashidze, SVN Vishwanathan, Tobias Petri, Kurt Mehlhorn, and Karsten Borgwardt. Efficient graphlet kernels for large graph comparison. In *Artificial intelligence and statistics*, pp. 488–495. PMLR, 2009.

Nino Shervashidze, Pascal Schweitzer, Erik Jan Van Leeuwen, Kurt Mehlhorn, and Karsten M Borgwardt. Weisfeiler-lehman graph kernels. *Journal of Machine Learning Research*, 12(9), 2011.

Arnab Sinha, Zhihong Shen, Yang Song, Hao Ma, Darrin Eide, Bo-June Hsu, and Kuansan Wang. An overview of microsoft academic service (mas) and applications. In *Proceedings of the 24th international conference on world wide web*, pp. 243–246, 2015.

Fan-Yun Sun, Jordan Hoffmann, Vikas Verma, and Jian Tang. Infograph: Unsupervised and semi-supervised graph-level representation learning via mutual information maximization. *arXiv preprint arXiv:1908.01000*, 2019.

Mengying Sun, Jing Xing, Huijun Wang, Bin Chen, and Jiayu Zhou. Mocl: Contrastive learning on molecular graphs with multi-level domain knowledge. *arXiv preprint arXiv:2106.04509*, 2021.

Susheel Suresh, Pan Li, Cong Hao, and Jennifer Neville. Adversarial graph augmentation to improve graph contrastive learning. *NeurIPS*, 34:15920–15933, 2021.

Shantanu Thakoor, Corentin Tallec, Mohammad Gheshlaghi Azar, Rémi Munos, Petar Veličković, and Michal Valko. Large-scale representation learning on graphs via bootstrapping. In *ICLR*, 2022.

Yonglong Tian, Chen Sun, Ben Poole, Dilip Krishnan, Cordelia Schmid, and Phillip Isola. What makes for good views for contrastive learning? *Advances in Neural Information Processing Systems*, 33:6827–6839, 2020.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Petar Veličković, William Fedus, William L Hamilton, Pietro Liò, Yoshua Bengio, and R Devon Hjelm. Deep graph infomax. *arXiv preprint arXiv:1809.10341*, 2018.

Tongzhou Wang and Phillip Isola. Understanding contrastive representation learning through alignment and uniformity on the hypersphere. In *International Conference on Machine Learning*, pp. 9929–9939. PMLR, 2020.

Jonathan Waring, Charlotta Lindvall, and Renato Umeton. Automated machine learning: Review of the state-of-the-art and opportunities for healthcare. *Artificial intelligence in medicine*, 104: 101822, 2020.

Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *ICML*, 2019.

Jun Xia, Lirong Wu, Jintao Chen, Bozhen Hu, and Stan Z Li. Simgrace: A simple framework for graph contrastive learning without data augmentation. In *Proceedings of the ACM Web Conference 2022*, pp. 1070–1079, 2022.

Yaochen Xie, Zhao Xu, Jingtun Zhang, Zhengyang Wang, and Shuiwang Ji. Self-supervised learning of graph neural networks: A unified review. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2022.

Dongkuan Xu, Wei Cheng, Dongsheng Luo, Haifeng Chen, and Xiang Zhang. Infogcl: Information-aware graph contrastive learning. *Advances in Neural Information Processing Systems*, 34, 2021.

Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. How powerful are graph neural networks? *arXiv preprint arXiv:1810.00826*, 2018.

Pinar Yanardag and SVN Vishwanathan. Deep graph kernels. In *Proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*, pp. 1365–1374, 2015.

Yihang Yin, Qingzhong Wang, Siyu Huang, Haoyi Xiong, and Xiang Zhang. Autogcl: Automated graph contrastive learning via learnable view generators. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pp. 8892–8900, 2022.

Zhitao Ying, Jiaxuan You, Christopher Morris, Xiang Ren, Will Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *Advances in neural information processing systems*, 31, 2018.

Yuning You, Tianlong Chen, Yongduo Sui, Ting Chen, Zhangyang Wang, and Yang Shen. Graph contrastive learning with augmentations. *NeurIPS*, 33:5812–5823, 2020.

Yuning You, Tianlong Chen, Yang Shen, and Zhangyang Wang. Graph contrastive learning automated. *arXiv preprint arXiv:2106.07594*, 2021.

Jure Zbontar, Li Jing, Ishan Misra, Yann LeCun, and Stéphane Deny. Barlow twins: Self-supervised learning via redundancy reduction. In *International Conference on Machine Learning*, pp. 12310–12320. PMLR, 2021.

Hengrui Zhang, Qitian Wu, Junchi Yan, David Wipf, and Philip S Yu. From canonical correlation analysis to self-supervised graph neural networks. *Advances in Neural Information Processing Systems*, 34:76–89, 2021.

Xin Zhang, Qiaoyu Tan, Xiao Huang, and Bo Li. Graph contrastive learning with personalized augmentation. *arXiv preprint arXiv:2209.06560*, 2022.

Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020.

Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Deep graph contrastive representation learning. *arXiv preprint arXiv:2006.04131*, 2020.

Yanqiao Zhu, Yichen Xu, Qiang Liu, and Shu Wu. An empirical study of graph contrastive learning. 2021a.

Yanqiao Zhu, Yichen Xu, Feng Yu, Qiang Liu, Shu Wu, and Liang Wang. Graph contrastive learning with adaptive augmentation. In *WWW*, pp. 2069–2080, 2021b.

A Experiment Setup

In this section, we introduce the datasets used in our experiments. Specifically, we adopt the following 6 popular node-level datasets and summarize their statistics in Table 4.

- **Cora**, and **PubMed**: They are two widely used citation network datasets (Sen et al., 2008). Nodes represent documents and edges denote citation links. Each node has a sparse bag-of-the-words feature vectors. Labels are defined as the academic topics.
- **Amazon-Computers** and **Amazon-Photo**: They are two networks of co-purchase relationships constructed from Amazon (McAuley et al., 2015). Nodes indicate goods and edges represent the co-purchase relationships of two products. Each node has a sparse bag-of-words feature encoding products reviews and is labeled with its category. They are widely used for node classification task. Nodes represent authors and edges indicate co-authorship relationships. Each node has a sparse bag-of-words feature based on paper keywords of the author. The task is to predict the most active research field of authors.
- **Coauthor-CS** and **Coauthor-Physics**: They are two academic networks, which represent co-authorship graphs based on the Microsoft Academic Graph from the KDD Cup 2016 challenge (Sinha et al., 2015).

Moreover, we also consider 9 graph-level benchmark datasets to verity the effectiveness of PerturbGCL on graph-learning task. Specifically, we use 5 social networks (COLLAB, REDDIT-BINARY, REDDIT-MULTI-5K, IMDB-BINARY, and GITHUB), and 2 molecules networks (NCI1 and MUTAG), and 2 bioinformatics networks (PROTEINS and DD) from the benchmark TUDdataset (Morris et al., 2020). Table 5 lists their statistics.

Table 4: Dataset statistics of node-level benchmarks.

Data	# Nodes	# Edges	# Features	Split ratio	# Classes
Cora	$2,708$	$5,429$	$1,433$	$85/5/15$	7
PubMed	$19,717$	$44,338$	500	$85/5/15$	3
Amazon-Computers	$13,752$	$245,861$	767	–	10
Amazon-Photo	$7,650$	$119,081$	745	–	8
Coauthor-CS	$18,333$	$81,894$	$6,805$	–	15
Coauthor-Physics	$34,493$	$247,962$	$8,415$	–	5

A.1 Setup for node classification

Dataset. We use 2 Planetoid graphs (Cora and PubMed), and 4 widely used datasets (Shchur et al., 2018) (Amazon-Computers, Amazon-Photo, Coauthor-CS, and Coauthor-Physics) for experiments. For Cora and PubMed, we follow the common semi-supervised practice (Kipf & Welling, 2016a) to generate train/val/test data splits without any modifications. For Amazon-Computers, Amazon-Photo, Coauthor-CS and Coauthor-Physics, since there are no data splits available, so similar to BGRL (Thakoor et al., 2022) and GBT (Bielak et al., 2022), we generate 20 random train/val/test splits (10%/10%/80%).

Competitors. To have a rigorous and comprehensive comparison with state-of-the-art methods, we compare PerturbGCL with 2 classical unsupervised models: DeepWalk (Perozzi et al., 2014) and GAE (Kipf & Welling, 2016b), 9 standard self-supervised models, DGI (Veličković et al., 2018), GRACE (Zhu et al., 2020), MVGRL (Hassani & Khasahmadi, 2020), GCA (Zhu et al., 2021b), BGRL (Thakoor et al., 2022), GBT (Bielak et al., 2022), InforGCL (Xu et al., 2021), CCA-SSG (Zhang et al., 2021), AFGRL (Lee et al., 2021), and one model perturbation method – SimGRACE (Xia et al., 2022). We also compare with supervised learning models including GCN (Kipf & Welling, 2016a) and GAT (Veličković et al., 2017). The results of baselines are quoted from (Zhang et al., 2021; Xu et al., 2021; Xia et al., 2022) if not specified.

Evaluation protocol. We follow the popular linear evaluation scheme to evaluate the performance of unsupervised models. Specifically, we first pre-train the model on the given graph without using ground-truth labels. Then, we freeze the parameters of the encoder and use it to generate node representations. After that, the generated node representations would be fed into a linear classification,

Table 5: Dataset statistics of graph-level benchmarks.

	$\mid \mathcal{G} \mid$	Avg.Nodes	Avg.Edges	$\#Label$
NCI1	$4,110$	29.87	32.30	2
PROTEINS	$1,113$	39.06	72.82	2
DD	$1,178$	284.32	715.66	2
MUTAG	188	17.93	19.79	2
COLLAB	$5,000$	74.49	$2,457.78$	3
IMDB-BINARY	$1,000$	19.77	96.53	2
REDDIT-BINARY	$2,000$	429.63	497.75	2
REDDIT-MULTI-5K	$4,999$	508.52	594.87	5
GITHUB	$12,725$	113.79	234.64	2

Table 6: Hyperparameters of PerturbGCL on node classification. We use GCN (Kipf & Welling, 2016a) as the backbone encoder.

	Cora	PubMed	Computer	Photo	CS	Phy
# layers	2	2	2	2	2	2
hidden dim	512	512	512	512	512	512
learning rate	0.005	0.0005	0.001	0.00001	0.00001	0.00001
propagation step	3	3	3	2	3	3
pruning ratio	0.5	0.7	0.7	0.9	0.9	0.9

i.e., a simple logistic regression model, to make the prediction for each node. It is worth noting that only nodes in the training set are used as supervision when training the classifier, and we report the accuracy results on testing nodes.

We implement PerturbGCL with PyTorch and use Adam optimizer to train the model. The graph encoder f_w is specified as a standard two-layer GCN model for all the datasets. We have two hyperparameters (pruning ratio s and random propagation step K) to tune. For each dataset, we search $K \in [1, 2, 3]$ and $s \in [0.5, 0.7, 0.9]$. To avoid randomness, we report the mean accuracy with a standard deviation through 10 random initialization. The detailed hyperparameter settings are summarized in Table 6.

A.2 Setup for Unsupervised graph classification

Dataset. For the unsupervised graph classification task, we adopt 8 benchmark datasets (NCI1, PROTEINS, DD, MUTAG, COLLAB, RDT-B, RDT-MSK, and IMDB-B) for experiments, following (You et al., 2020). There are 20 epochs pre-training under the naive-strategy. After the

Competitors. We compare with the kernel-based methods like graphlet kernel (GL) (Shervashidze et al., 2009), Weisfeiler-Lehman sub-tree kernel (WL) (Shervashidze et al., 2011), and deep graph kernel (DGK) (Yanardag & Vishwanathan, 2015), and other unsupervised graph representation meodels like node2vec (Grover & Leskovec, 2016), sub2vec (Adhikari et al., 2018), graph2vec (Narayanan et al., 2017), as well as the state-of-the-art GCL methods like MVGRL (Hassani & Khasahmadi, 2020), InforGraph (Sun et al., 2019), GraphCL (You et al., 2020), JOAO (You et al., 2021), and SimGRACE (Xia et al., 2022).

Evaluation protocol. Following GraphCL (You et al., 2020), contrastively train the representation model using unlabeled graph data, and then fix the representation model and train a downstream classifier using labeled data. Specifically, we adopt SVM as the classifier and perform 10-fold cross validation. For each fold, we employ 90% of the data as the labeled data for training and the remaining 10% as the labeled testing data. To avoid randomness, we repeatedly run experiments for 5 times and report the averaged results.

Following GraphCL (You et al., 2020), we use GIN (Xu et al., 2018) as the GNN backbone, and also search the best K and s from $\{1, 2, 3\}$ and $\{0.5, 0.7, 0.9\}$, respectively. Table 7 reports the parameter configurations for all datasets.

Table 7: Hyperparameters of PerturbGCL on unsupervised graph classification. We use GIN (Xu et al., 2018) as the backbone encoder.

	NCI1	PROTEINS	DD	MUTAG	COLLAB	RDT-B	RDT-M5K	IMDB-B
# layers	3	3	3	3	3	3	3	3
hidden dim	32	32	32	32	32	32	32	32
learning rate	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
propagation step	2	2	3	2	3	2	2	2
pruning ratio	0.9	0.7	0.9	0.7	0.9	0.9	0.5	0.5

Table 8: Hyperparameters of PerturbGCL on semi-supervised graph classification. We use Res-GCN (Chen et al., 2019) as the backbone encoder.

	NCI1	PROTEINS	DD	MUTAG	COLLAB	RDT-B	RDT-M5K	GITHUB
# layers	5	5	5	3	5	5	3	3
hidden dim	128	128	128	128	128	128	128	128
learning rate	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
propagation step	2	2	3	2	3	2	2	2
pruning ratio	0.5	0.7	0.9	0.7	0.9	0.7	0.7	0.7

A.3 Setup for Semi-Supervised graph classification

Dataset. We perform semi-supervised graph classification task 7 popular benchmark datasets (NCI1, PROTEINS, DD, COLLAB, RDT-B, RDT-M5K, and GITHUB) from TUDataset (Morris et al., 2020). There are 100 epochs pre-training under the default setting.

Competitors. We compare with unsupervised graph representation meodels: GAE (Kipf & Welling, 2016b), Infomax (DGI) (Veličković et al., 2018), and ContextPred (Hu et al., 2019), and other state-of-the-art GCL methods like InforGraph (Sun et al., 2019), GraphCL (You et al., 2020), JOAO (You et al., 2021), and SimGRACE (Xia et al., 2022).

Evaluation protocol. We employ a 10-fold cross validation on each dataset. For each fold, we use 80% of the data as the unlabeled data, 10% as labeled training data, and 10% as labeled testing data. For the augmentation only (Augmentations) experiments, we only perform 30 epochs of supervised training with augmentations using labeled data.

Following GraphCL (You et al., 2020), we use ResGCN (Chen et al., 2019) as the GNN backbone, and also search the best K and s from $\{1, 2, 3\}$ and $\{0.5, 0.7, 0.9\}$, respectively. Table 8 reports the parameter configurations for all datasets.

Algorithm 1 PerturbGCL on node level task

```
1: Input: Original graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, GNN encoder $f_w(\cdot)$ with weight $W$, projection head
   $h(\cdot)$, the maximum propagation step $K$, and pruning ratio $s$
2: Initialize the encoder $f_w(\cdot)$ and set mask indicator to ones.
3: for iterate 1, 2, ... times until convergence do
4:     Sample the random propagation steps $k', k''$ from the uniform distribution $U(1, K)$
5:     Conduct weight pruning to update the mask indicator $m_w$
6:     Compute the target representation $\mathbf{h}_v$ according to 1 by performing $k'$ times of $g()$
7:     Get $\mathbf{z}_v = h(\mathbf{h}_v)$ according to 2
8:     Compute the perturbed representation $\mathbf{h}_v^+$ according to 1 by performing $k''$ times of $g()$ and
       using masked weight $W \odot m_w$
9:     Get $\mathbf{z}_v^+ = h(\mathbf{h}_v)$
10:    define $\mathcal{L}_{CL} = \frac{1}{|V|} \sum_{v \in \mathcal{V}} - \log \frac{\exp(\text{sim}(\mathbf{z}_v, \mathbf{z}_v^+)/\tau)}{\exp(\text{sim}(\mathbf{z}_v, \mathbf{z}_v^+)/\tau) + \sum_{u \in \mathcal{V}, u \neq v} \exp(\text{sim}(\mathbf{z}_v, \mathbf{z}_u^-)/\tau)}$ according to 3
11:    Optimize $f_w(\cdot), h(\cdot)$ to minimize $\mathcal{L}_{CL}$
12: end for
13: return the pre-trained GNN encoder $f_w(\cdot)$
```

Algorithm 2 PerturbGCL on graph level task

```
1:  Input: Original graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathbf{X})$, GNN encoder $f_w(\cdot)$ with weight $W$, projection head
    $h(\cdot)$, the maximum propagation step $K$, and pruning ratio $s$
2:  Initialize the encoder $f_w(\cdot)$ and set mask indicator to ones.
3:  for iterate 1, 2, ... times until convergence do
4:      for sampling batches $\mathcal{G}_n$ from $\mathcal{G}$, where $n = 1, 2, ..., N$ do
5:          Sample the random propagation steps $k', k''$ from the uniform distribution $U(1, K)$
6:          Conduct weight pruning to update the mask indicator $m_w$
7:          Compute the target representation $\mathbf{h}_v$ according to 1 by performing $k'$ times of $g()$
8:          Get $\mathbf{z}_{\mathcal{G}} = \text{READOUT}(h(\mathbf{h}_v)_{v \in \mathcal{G}_n})$ according to 2
9:          Compute the perturbed representation $\mathbf{h}_v^+$ according to 1 by performing $k''$ times of $g()$
            and using masked weight $W \odot m_w$
10:         Get $\mathbf{z}_{\mathcal{G}_n}^+ = \text{READOUT}(h(\mathbf{h}_v^+)_{v \in \mathcal{G}_n})$
11:         define $\mathcal{L}_{CL} = \frac{1}{|\mathcal{G}|} \sum_{\mathcal{G}_n \in \mathcal{G}} -\log \frac{\exp(\text{sim}(\mathbf{z}_{\mathcal{G}_n}, \mathbf{z}_{\mathcal{G}_n}^+)/\tau)}{\exp(\text{sim}(\mathbf{z}_{\mathcal{G}_n}, \mathbf{z}_{\mathcal{G}_n}^+)/\tau) + \sum_{u \in \mathcal{G}, u \neq v} \exp(\text{sim}(\mathbf{z}_{\mathcal{G}_n}, \mathbf{z}_u^-)/\tau)}$ accord-
            ing to 3
12:         Optimize $f_w(\cdot), p(\cdot)$ to minimize $\mathcal{L}_{CL}$
13:     end for
14: end for
15: return The pre-trained GNN encoder $f(\cdot)$
```

B Details for Toy Example

To verify the limitation of SimGRACE (Xia et al., 2022) on handling perturbation created by data augmentation. We select three popular data augmentation based baselines: GRACE (Zhu et al., 2020), BGRL (Thakoor et al., 2022), and CCA-SSG (Zhang et al., 2021). To measure the qualify of the representation models on learning representations for input data, we adopt the widely used *alignment* and *uniformity* metrics (Wang & Isola, 2020) for quantitative analysis. According to (Wang & Isola, 2020), both metrics are the smaller the better.

Evaluation setting. For all methods, we first pre-train them according to their own configurations on PubMed, Amazon-Photo, and Coauthor-CS datasets. Then, we use data augmentation strategies to construct two perturbed views. Specifically, we following this detailed empirical study (Zhu et al., 2021a) and adopt *edge perturbation* and *attribute masking* as the default perturbation function on the input graph. To have a fair comparison, we fix the random seed and generated two shared perturbed graphs, and then feed the two views into BGRL, GRACE, CCA-SSG, and SimGRACE to obtain node representations for all nodes in the graph. After that, we use the obtained node representations of two views to compute the *alignment* and *uniformity* according to Eq. 5. WE repeat the process for 10 times and report the averaged results in Figure 1.

Table 9: Test accuracy on benchmark datasets in TUdatasets in terms of semi-supervised graph classification.

Dataset	NCI1	PROTEINS	DD	COLLAB	RDT-B	RDT-M5K	GITHUB	A.R. ↓
No pre-train	73.72 ± 0.24	70.40 ± 1.54	73.56 ± 0.41	73.71 ± 0.27	86.63 ± 0.27	51.33 ± 0.44	60.87 ± 0.17	9.86
Augmentations	73.59 ± 0.32	70.29 ± 0.64	74.30 ± 0.81	74.19 ± 0.13	87.74 ± 0.39	52.01 ± 0.20	60.91 ± 0.32	9.00
GAE	74.36 ± 0.24	70.51 ± 0.17	74.54 ± 0.68	75.09 ± 0.19	87.69 ± 0.40	53.58 ± 0.13	63.89 ± 0.52	7.14
InfoGraph	74.86 ± 0.26	72.27 ± 0.40	75.78 ± 0.34	73.76 ± 0.29	88.66 ± 0.95	53.61 ± 0.31	65.21 ± 0.88	4.71
ContextPred	73.00 ± 0.30	70.23 ± 0.63	74.66 ± 0.51	73.60 ± 0.37	84.76 ± 0.52	51.23 ± 0.84	–	10.50
Infomax	74.86 ± 0.26	72.27 ± 0.40	75.78 ± 0.34	73.76 ± 0.29	88.66 ± 0.95	53.61 ± 0.31	65.21 ± 0.88	5.71
GraphCL	74.63 ± 0.25	$\mathbf{74.17 \pm 0.34}$	$\mathbf{76.17 \pm 1.37}$	74.23 ± 0.21	$\mathbf{89.11 \pm 0.19}$	52.55 ± 0.45	65.81 ± 0.79	3.86
JOAO	74.48 ± 0.25	72.13 ± 0.92	75.69 ± 0.67	$\mathbf{75.30 \pm 0.32}$	88.14 ± 0.25	52.83 ± 0.54	65.00 ± 0.30	6.14
JOAOv2	74.86 ± 0.39	73.31 ± 0.48	75.81 ± 0.73	$\mathbf{75.53 \pm 0.18}$	88.79 ± 0.65	52.71 ± 0.28	66.60 ± 0.60	3.57
SimGRACE	74.60 ± 0.41	$\mathbf{74.03 \pm 0.51}$	$\mathbf{76.48 \pm 0.52}$	74.74 ± 0.28	88.86 ± 0.62	53.97 ± 0.64	65.33 ± 0.35	3.29
PerturbGCL	$\mathbf{75.23 \pm 0.52}$	$\mathbf{74.11 \pm 0.42}$	$\mathbf{76.65 \pm 0.57}$	74.50 ± 0.41	88.69 ± 0.63	$\mathbf{55.39 \pm 0.12}$	$\mathbf{68.40 \pm 0.10}$	2.14

C More Contrastive loss functions

Although InfoNCE (You et al., 2020) (illustrated in Eq. 3) is the widely used contrastive objective in learning GCL models, some other training objectives have been proposed recently, such as Barlow Twins (Bielak et al., 2021), Bootstrap (Thakoor et al., 2022). Different from InfoNCE, the other two contrastive objectives are negative-sample free, so they can avoid the efforts to identify informative negative samples during the training. Specifically, the core idea of Bootstrap function is to maximize the difference between the positive pairs, defined as:

$$\mathcal{L} = -\frac{2}{N}\sum_{v\in\mathcal{V}}^{N-1}\frac{\mathbf{z}_v\mathbf{h}_v^+}{\|\mathbf{z}_v\|\,\|\mathbf{h}_v^+\|}. \tag{7}$$

Here $\mathbf{h}_v^+$ is the hidden representation encoded by GNN encoder, and $\mathbf{z}_v = h(\mathbf{h}_v^+)$, where $h(\cdot)$ is the prediction head. In our cases, since we take the original graph as input, so we do not have symmetric define of the loss function as done in (Thakoor et al., 2022). This might be the reason why our model PerturbGCL performs not good using this objective.

Barlow Twins (Bielak et al., 2021) is a recent endeavor to reduce the usage of negative samples. This objective is originally proposed in image domain by (Zbontar et al., 2021). The general idea of Barlow Twins is to minimize the redundancy in the hidden dimension. Specifically, given the hidden representation of two views ($\mathbf{Z}$ and $\mathbf{Z}^+$), it first compute the empirical cross-correlation matrix $\mathcal{C} \in \mathbb{R}^{D\times D}$ as below:

$$\mathcal{C}_{i,j} = \frac{\sum_n \mathbf{Z}_{n,i}\mathbf{Z}_{n,j}^+}{\sqrt{\sum_n \left(\mathbf{Z}_{n,i}\right)^2}\sqrt{\sum_n \left(\mathbf{Z}_{n,j}^+\right)^2}}, \tag{8}$$

where n is the the batch indexes and i, j are the indexes of embeddings. The cross-correlation matrix $\mathcal{C}$ is optimized to be equal to the identity matrix. To be specific, it is composed of two parts: 1) the invariance term and 2) the redundancy reduction term. The first one forces the on diagonal elements $\mathcal{C}_{i,i}$ to be equal to one, hence making the embeddings invariant to the applied augmentations. The second term optimizes the off-diagonal elements $\mathcal{C}_{i,j}$ to be equal to zero – this results in decorrelated components of the embedding vectors. Formally, the loss function $\mathcal{L}_{BT}$ is computed by:

$$\mathcal{L}_{BT} = \sum_i (1-\mathcal{C}_{i,i})^2 + \lambda\sum_i\sum_{j\neq i}\mathcal{C}_{i,j}^2. \tag{9}$$

The $\lambda > 0$ parameter defines the trade-off between the invariance and redundancy reduction terms. In our experiments, we set $\lambda = \frac{1}{D}$ following (Bielak et al., 2021).

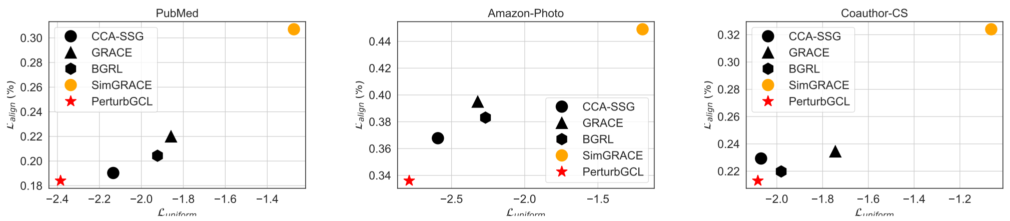

Figure 6: The *alignment* and *uniformity* plot for BGRL (Thakoor et al., 2022), SimGRACE (Xia et al., 2022), GRACE (Zhu et al., 2020), CCA-SSG (Zhang et al., 2021), and our PerturbGCL on the same perturbed graphs generated by data augmentation. *Black circles* (●) indicate the baselines. *Orange circles* (🟠) represent the performance of SimGRACE. *Red starts* (★) are the results of PerturGCL.

D Efficiency analysis

To evaluate the efficiency of the proposed PerturbGCL, we compare it with two strong GCL baselines: BGRL (Thakoor et al., 2022) on node classification data and GraphCL (You et al., 2020) on graph classification data. It is worth to note that since BGRL and GraphCL require to search for the optimal augmentation strategies via trial-and-error, which is super expensive in practice. To simplify the comparison, we fix their optimal augmentations and only record the running time on the optimal augmentations. For graph-level datastes, GraphCL and PerturbGCL use 3 GIN layers with hidden dimension 32 as the backbone encoder. The propagation step and pruning ratio of PerturbGCL are set as $K = 2$ and $s = 0.7$. For node-level datasets, BGRL and PerturbGCL use 2 GCN layers with hidden dimension 512 as the backbone encoder. The propagation step and pruning ratio of PerturbGCL are set as $K = 3$ and $s = 0.9$. We conduct experiments on a server with AMD EPYC 7282 16-Core processors, 252 GB memory, and one GeForce RTX 2080 Ti GPUs (24GB). To avoid randomness, the reported results in Table 3 are the averaged performance over 100 training epochs.

Except the improvement in Table 3, another thing we want to mention is that the practical speedup of PerturbGCL should be significantly higher than the results shown in Table 3. This is mainly because the data augmentation based GCL methods require a lot of efforts to search for the best augmentation strategies, such as the best augmentation types and their corresponding perturbation ratios.

E More Results

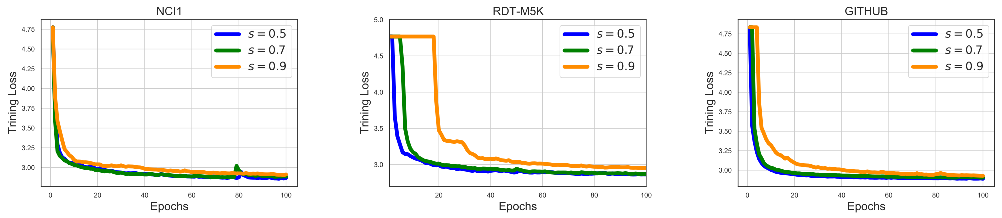

Figure 7: Empirical training curves of PerturbGCL on graph benchmarks with different pruning ratios s.

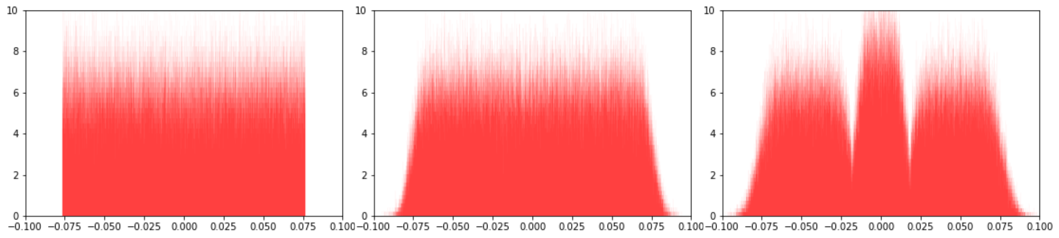

Figure 8: Visualization of second layer weight distribution during training process(from left to right: initial weights, PerturbGCL w/o. *weightPrune*, and PerturbGCL) on Coauthor-CS. The x-axis indicates weight values and y-axis is the corresponding count.

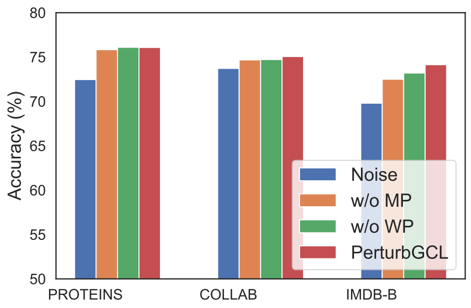

Figure 9: Ablation study of PerturbGCL on node benchmarks.

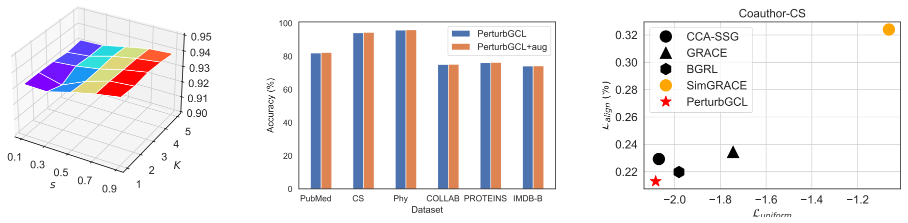

Figure 10: **Left:** Hyperparameter Analysis on Coauthor-CS. **Middle:** PerturbGCL with data augmentation. **Right:** The *alignment* and *uniformity* results of PerturbGCL.